zepto.JS 

目錄

Zepto.js API 中文文档 1.1.6

Zepto.js



Zepto是一个轻量级的针对现代高级浏览器的**JavaScript**库， 它与jquery有着类似的**api**。 如果你会用jquery，那么你也会用zepto。

Zepto的设计目的是提供 **jQuery** 的类似的**API**，但并不是100%覆盖 jQuery 。**Zepto**设计的目的是有一个5-10k的通用库、下载并快速执行、有一个熟悉通用的API，所以你能把你主要的精力放到应用开发上。

Zepto 是一款开源软件，它可以被开发者和商业发布。 MIT license.

Tweet

Download Zepto

默认构建包含以下模块： *Core, Ajax, Event, Form, IE*.

Zepto v1.0 默认捆绑了Effects, iOS3, 和 Detect 模块。 请参阅下面的 可选模块(optional modules)。

- zepto.js v1.1.6 (for development) – *54.6k uncompressed, lots of comments*
- zepto.min.js v1.1.6 (for production) – *9.1k when gzipped*

Or **grab the latest version on GitHub**.

用一个script标签引入Zepto到你的页面的底部：

```
...
<script src=zepto.min.js></script>
</body>
</html>
```

如果 `$` 变量尚未定义，Zepto只设置全局变量 `$` 指向它本身。 没有 `Zepto.noConflict` 方法。

如果你需要支持旧的浏览器，如**Internet Explorer 9**或以下，你可以退回到**jQuery**的**1.x**。

浏览器支持

初级 (100% 支持)

- Safari 6+ (Mac)
- Chrome 30+ (Windows, Mac, Android, iOS, Linux, Chrome OS)
- Firefox 24+ (Windows, Mac, Android, Linux, Firefox OS)
- iOS 5+ Safari
- Android 2.3+ Browser
- Internet Explorer 10+ (Windows, Windows Phone)

次要目标（完全或大部分支持）

- iOS 3+ Safari
- Chrome <30
- Firefox 4+
- Safari <6
- Android Browser 2.2
- Opera 10+
- webOS 1.4.5+ Browser
- BlackBerry Tablet OS 1.0.7+ Browser
- Amazon Silk 1.0+
- Other WebKit-based browsers/runtimes

需要注意的是*Zepto*的一些可选功能是专门针对移动端浏览器的；因为它的最初目标在移动端提供一个精简的类似*jquery*的*js*库。

在浏览器上(Safari、Chrome和Firefox)上开发页面应用或者构建基于html的web-view本地应用，你如PhoneGap，使用Zepto是一个不错的选择。

总之，Zepto希望在所有的现代浏览器中作为一种基础环境来使用。Zepto不支持旧版本的Internet Explorer浏览器(<10)。

手动建立Zepto

`zepto.js` 和 `zepto.min.js` 提供以上使用方式。 然而，为了更好的程序效果和自由性，可以在使用Zepto源码构建Zepto.js和zepto.min.js的时候选择模块并作测试， 使用UglifyJS根据你的需要来生成(当服务端开启gzipped后，最精简的代码)代码。

关于如何建立Zepto的the README，包含运行测试和补丁。

Zepto 模块

module	default	description
zepto	yes	核心模块；包含许多方法
event	yes	通过 `on()` & `off()` 处理事件
ajax	yes	XMLHttpRequest 和 JSONP 实用功能
form	yes	序列化 & 提交web表单
ie	yes	增加支持桌面的Internet Explorer 10+和Windows Phone 8。
detect		提供 `$.os` 和 `$.browser` 消息
fx		The `animate()` 方法
fx_methods		以动画形式的 `show`， `hide`， `toggle`，和 `fade*()` 方法.
assets		实验性支持从DOM中移除image元素后清理iOS的内存。
data		一个全面的 `data()` 方法, 能够在内存中存储任意对象。
deferred		提供 `$.Deferred` promises API. 依赖"callbacks" 模块. 当包含这个模块时候, `$.ajax()` 支持promise接口链式的回调。
callbacks		为"deferred"模块提供 `$.Callbacks` 。
selector		实验性的支持 jQuery CSS 表达式 实用功能，比如 `$('div:first')` 和 `el.is(':visible')` 。
touch		在触摸设备上触发tap– 和 swipe– 相关事件。这适用于所有的 `touch` (iOS, Android)和 `pointer` 事件(Windows Phone)。
gesture		在触摸设备上触发 pinch 手势事件。
stack		提供 `andSelf` & `end()` 链式调用方法
ios3		String.prototype.trim 和 Array.prototype.reduce 方法 (如果他们不存在) ，以兼容 iOS 3.x.

创建插件

可以通过添加方法作为 `$.fn` 的属性来写插件：

```
;(function($){
  $.extend($.fn, {
    foo: function(){
      // `this` refers to the current Zepto collection.
      // When possible, return the Zepto collection to allow chaining.
      return this.html('bar')
    }
  })
})(Zepto)
```

为了更好开始开发插件，先看下source of Zepto's core module，并确认读过coding style guidelines

核心方法

$()

```
$(selector, [context])   => collection
$(<Zepto collection>)   => same collection
$(<DOM nodes>)   => collection
$(htmlString)   => collection
$(htmlString, attributes)   => collection v1.0+
Zepto(function($){ ... })
```

通过执行css选择器，包装dom节点，或者通过一个html字符串创建多个元素 来创建一个Zepto集合对象。

Zepto集合是一个类似数组的对象，它具有链式方法来操作它指向的DOM节点，除了$(`Zepto`)对象上的直接方法外(如 `$.extend`)，文档对象中的所有方法都是集合方法。

如果选择器中存在content参数(css选择器，dom，或者Zepto集合对象)，那么只在所给的节点背景下进行css选择器；这个功能和使用 `$(context).find(selector)` 是一样的。

当给定一个html字符串片段来创建一个dom节点时。也可以通过给定一组属性映射来创建节点。最快的创建但元素，使用 `<div>` 或 `<div/>` 形式。

当一个函数附加在 `DOMContentLoaded` 事件的处理流程中。如果页面已经加载完毕，这个方法将会立即被执行。

```
$('div')  //=> 所有页面中得div元素
$('#foo') //=> ID 为 "foo" 的元素

// 创建元素:
$("<p>Hello</p>") //=> 新的p元素
// 创建带有属性的元素:
$("<p />", { text:"Hello", id:"greeting", css:{color:'darkblue'} })
//=> <p id=greeting style="color:darkblue">Hello</p>

// 当页面ready的时候，执行回调:
Zepto(function($){
  alert('Ready to Zepto!')
})
```

不支持jQuery CSS 扩展， 然而，可选的“selector”模块有限提供了支持几个最常用的伪选择器，而且可以被丢弃，与现有的代码或插件的兼容执行。

如果 `$` 变量尚未定义，Zepto只设置了全局变量 `$` 指向它本身。允许您同时使用的Zepto和有用的遗留代码，例如，prototype.js。只要首先加载Prototype，Zepto将不会覆盖Prototype的 `$` 函数。Zepto将始终设置全局变量 `Zepto` 指向它本身。

$.camelCase v1.0+

```
$.camelCase(string)   => string
```

将一组字符串变成“骆驼”命名法的新字符串，如果该字符已经是“骆驼”命名法，则不变化。

```
$.camelCase('hello-there') //=> "helloThere"
$.camelCase('helloThere')  //=> "helloThere"
```

$.contains v1.0+

```
$.contains(parent, node)   => boolean
```

检查父节点是否包含给定的dom节点，如果两者是相同的节点，则返回 `false` 。

$.each

```
$.each(collection, function(index, item){ ... })   => collection
```

遍历数组元素或以key-value值对方式遍历对象。回调函数返回 `false` 时停止遍历。

```
$.each(['a', 'b', 'c'], function(index, item){
  console.log('item %d is: %s', index, item)
})

var hash = { name: 'zepto.js', size: 'micro' }
$.each(hash, function(key, value){
  console.log('%s: %s', key, value)
})
```

$.extend

```
$.extend(target, [source, [source2, ...]])   => target
$.extend(true, target, [source, ...])   => target v1.0+
```

通过源对象扩展目标对象的属性，源对象属性将覆盖目标对象属性。

默认情况下为，复制为浅拷贝（浅复制）。如果第一个参数为true表示深度拷贝（深度复制）。

```
var target = { one: 'patridge' },
    source = { two: 'turtle doves' }

$.extend(target, source)
//=> { one: 'patridge',
//     two: 'turtle doves' }
```

$.fn

`Zepto.fn` 是一个对象，它拥有Zepto对象上所有可用的方法，如 `addClass()`， `attr()`，和其它方法。在这个对象添加一个方法，所有的Zepto对象上都能用到该方法。

这里有一个实现 Zepto 的 `empty()` 方法的例子：

```
$.fn.empty = function(){
  return this.each(function(){ this.innerHTML = '' })
}
```

$.grep v1.0+

```
$.grep(items, function(item){ ... })   => array
```

获取一个新数组，新数组只包含回调函数中返回 ture 的数组项。

```
$.grep([1,2,3],function(item){
    return item > 1
});//=>[2,3]
```

$.inArray v1.0+

```
$.inArray(element, array, [fromIndex])   => number
```

返回数组中指定元素的索引值（愚人码头注：以0为基数），如果没有找到该元素则返回 `-1`。

愚人码头注：`[fromIndex]` 参数可选，表示从哪个索引值开始向后查找。

```
$.inArray("abc",["bcd","abc","edf","aaa"]);//=>1

$.inArray("abc",["bcd","abc","edf","aaa"],1);//=>1

$.inArray("abc",["bcd","abc","edf","aaa"],2);//=>-1
```

$.isArray

```
$.isArray(object)   => boolean
```

如果object是array，则返回ture。

$.isFunction

```
$.isFunction(object)   => boolean
```

如果object是function，则返回ture。

$.isPlainObject v1.0+

```
$.isPlainObject(object)   => boolean
```

测试对象是否是“纯粹”的对象，这个对象是通过 对象常量（"{}"） 或者 `new Object` 创建的，如果是，则返回true。

```
$.isPlainObject({})         // => true
$.isPlainObject(new Object) // => true
$.isPlainObject(new Date)   // => false
$.isPlainObject(window)     // => false
```

$.isWindow v1.0+

```
$.isWindow(object)   => boolean
```

如果object参数是否为一个window对象，那么返回true。这在处理iframe时非常有用，因为每个iframe都有它们自己的window对象，使用常规方法 `obj === window` 校验这些objects的时候会失败。

$.map

```
$.map(collection, function(item, index){ ... })   => collection
```

通过遍历集合中的元素，返回通过迭代函数的全部结果，（愚人码头注：一个新数组）`null` 和 `undefined` 将被过滤掉。

```
$.map([1,2,3,4,5],function(item,index){
        if(item>1){return item*item;}
});
// =>[4, 9, 16, 25]

$.map({"yao":1,"tai":2,"yang":3},function(item,index){
    if(item>1){return item*item;}
});
// =>[4, 9]
```

$.parseJSON v1.0+

```
$.parseJSON(string)  ? object
```

原生 `JSON.parse` 方法的别名。（愚人码头注：接受一个标准格式的 JSON 字符串，并返回解析后的 JavaScript 对象。）

$.trim v1.0+

```
$.trim(string)   => string
```

删除字符串首尾的空白符。类似String.prototype.trim()。

$.type v1.0+

```
$.type(object)   => string
```

获取JavaScript 对象的类型。可能的类型有： `null` `undefined` `boolean` `number` `string` `function` `array` `date` `regexp` `object` `error` 。

对于其它对象，它只是简单报告为“object”，如果你想知道一个对象是否是一个javascript普通对象，使用 isPlainObject。

add

```
add(selector, [context])   => self
```

添加元素到当前匹配的元素集合中。如果给定content参数，将只在content元素中进行查找，否则在整个document中查找。

```
<ul>
    <li>list item 1</li>
    <li>list item 2</li>
    <li>list item 3</li>
</ul>
<p>a paragraph</p>

<script type="text/javascript">
    $('li').add('p').css('background-color', 'red');
</script>
```

addClass

```
addClass(name)   => self
addClass(function(index, oldClassName){ ... })   => self
```

为每个匹配的元素添加指定的class类名。多个class类名使用空格分隔。

after

```
after(content)   => self
```

在每个匹配的元素后插入内容（愚人码头注：外部插入）。内容可以为html字符串，dom节点，或者节点组成的数组。

```
$('form label').after('<p>A note below the label</p>')
```

append

```
append(content)   => self
```

在每个匹配的元素末尾插入内容（愚人码头注：内部插入）。内容可以为html字符串，dom节点，或者节点组成的数组。

```
$('ul').append('<li>new list item</li>')
```

appendTo

```
appendTo(target)   => self
```

将匹配的元素插入到目标元素的末尾（愚人码头注：内部插入）。这个有点像 append，但是插入的目标与其相反。

```
$('<li>new list item</li>').appendTo('ul')
```

attr

```
attr(name)   => string
attr(name, value)   => self
attr(name, function(index, oldValue){ ... })   => self
attr({ name: value, name2: value2, ... })   => self
```

读取或设置dom的属性。如果没有给定value参数，则读取对象集合中第一个元素的属性值。当给定了value参数。则设置对象集合中所有元素的该属性的值。当value参数为 `null` ，那么这个属性将被移除(类似removeAttr)，多个属性可以通过对象键值对的方式进行设置。

要读取DOM的属性如 `checked` 和 `selected` , 使用 prop。

```
var form = $('form')
form.attr('action')             //=> 读取值
form.attr('action', '/create')  //=> 设置值
form.attr('action', null)       //=> 移除属性

// 多个属性:
form.attr({
  action: '/create',
  method: 'post'
})
```

before

```
before(content)   => self
```

在匹配每个元素的前面插入内容（愚人码头注：外部插入）。内容可以为html字符串，dom节点，或者节点组成的数组。

```
$('table').before('<p>See the following table:</p>')
```

children

```
children([selector])   => collection
```

获得每个匹配元素集合元素的直接子元素，如果给定selector，那么返回的结果中只包含符合css选择器的元素。

```
$('ol').children('*:nth-child(2n)')
//=> every other list item from every ordered list
```

clone v1.0+

```
clone()   => collection
```

通过深度克隆来复制集合中的所有元素。

此方法不会将数据和事件处理程序复制到新的元素。这点和jquery中利用一个参数来确定是否复制数据和事件处理不相同。

closest

```
closest(selector, [context])   => collection
closest(collection)   => collection v1.0+
closest(element)   => collection v1.0+
```

从元素本身开始，逐级向上级元素匹配，并返回最先匹配selector的元素。如果给定context节点参数，那么只匹配该节点的后代元素。这个方法与 `parents(selector)` 有点相像，但它只返回最先匹配的祖先元素。

如果参数是一个Zepto对象集合或者一个元素，结果必须匹配给定的元素而不是选择器。

```
var input = $('input[type=text]')
input.closest('form')
```

concat

```
concat(nodes, [node2, ...])   => self
```

添加元素到一个Zepto对象集合形成一个新数组。如果参数是一个数组，那么这个数组中的元素将会合并到Zepto对象集合中。

这是一个Zepto提供的方法，不是jquey的API 。

contents v1.0+

```
contents()   => collection
```

获得每个匹配元素集合元素的子元素，包括文字和注释节点。（愚人码头注：.contents() 和.children()方法类似，只不过前者包括文本节点以及jQuery对象中产生的HTML元素。）

css

```
css(property)   => value
css([property1, property2, ...])   => object v1.1+
css(property, value)   => self
css({ property: value, property2: value2, ... })   => self
```

读取或设置DOM元素的css属性。当value参数不存在的时候，返回对象集合中第一个元素的css属性。当value参数存在时，设置对象集合中每一个元素的对应css属性。

多个属性可以通过传递一个属性名组成的数组一次性获取。多个属性可以利用对象键值对的方式进行设置。

当value为空(空字符串，`null` 或 `undefined`)，那个css属性将会被移出。当value参数为一个无单位的数字，如果该css属性需要单位，“px”将会自动添加到该属性上。

```
var elem = $('h1')
elem.css('background-color')          // read property
elem.css('background-color', '#369')  // set property
elem.css('background-color', '')      // remove property

// set multiple properties:
elem.css({ backgroundColor: '#8EE', fontSize: 28 })

// read multiple properties:
elem.css(['backgroundColor', 'fontSize'])['fontSize']
```

data

```
data(name)   => value
data(name, value)   => self
```

读取或写入dom的 `data-*` 属性。行为有点像 attr ，但是属性名称前面加上 `data-` 。

当读取属性值时，会有下列转换：v1.0+

- “true”, “false”, and “null” 被转换为相应的类型；
- 数字值转换为实际的数字类型；

- JSON值将会被解析，如果它是有效的JSON；
- 其它的一切作为字符串返回。

  Zepto 基本实现`data()`只能存储字符串。如果你要存储任意对象，请引入可选的“data”模块到你构建的Zepto中。

each

```
each(function(index, item){ ... })   => self
```

遍历一个对象集合每个元素。在迭代函数中，`this`关键字指向当前项(作为函数的第二个参数传递)。如果迭代函数返回 `false`，遍历结束。

```
$('form input').each(function(index){
  console.log('input %d is: %o', index, this)
})
```

empty

```
empty()   => self
```

清空对象集合中每个元素的DOM内容。

eq

```
eq(index)   => collection
```

从当前对象集合中获取给定索引值（愚人码头注：以0为基数）的元素。

```
$('li').eq(0)   //=> only the first list item
$('li').eq(-1)  //=> only the last list item
```

filter

```
filter(selector)   => collection
filter(function(index){ ... })   => collection v1.0+
```

过滤对象集合，返回对象集合中满足css选择器的项。如果参数为一个函数，函数返回有实际值得时候，元素才会被返回。在函数中，`this`关键字指向当前的元素。

与此相反的功能，查看not.

find

```
find(selector)   => collection
find(collection)   => collection v1.0+
find(element)   => collection v1.0+
```

在当对象前集合内查找符合CSS选择器的每个元素的后代元素。

如果给定Zepto对象集合或者元素，过滤它们，只有当它们在当前Zepto集合对象中时，才回被返回。

```
var form = $('#myform')
form.find('input, select')
```

first

```
first()   => collection
```

获取当前对象集合中的第一个元素。

```
$('form').first()
```

forEach

```
forEach(function(item, index, array){ ... }, [context])
```

遍历对象集合中每个元素，有点类似 each，但是遍历函数的参数不一样，当函数返回 `false` 的时候，遍历不会停止。

这是一个Zepto提供的方法，不是jquery的API。

get

```
get()   => array
get(index)   => DOM node
```

从当前对象集合中获取所有元素或单个元素。当index参数不存在的时，以普通数组的方式返回所有的元素。当指定index时，只返回该置的元素。这点与eq不同，该方法返回的是DOM节点，不是Zepto对象集合。

```
var elements = $('h2')
elements.get()   //=> get all headings as an array
elements.get(0)  //=> get first heading node
```

has v1.0+

```
has(selector)   => collection
has(node)   => collection
```

判断当前对象集合的子元素是否有符合选择器的元素，或者是否包含指定的DOM节点，如果有，则返回新的对象集合，该对象过滤掉不含有选择器匹配元素或者不含有指定DOM节点的对象。

```
$('ol > li').has('a[href]')
//=> get only LI elements that contain links
```

hasClass

```
hasClass(name)   => boolean
```

检查对象集合中是否有元素含有指定的class。

```
<ul>
    <li>list item 1</li>
    <li class="yaotaiyang">list item 2</li>
    <li>list item 3</li>
</ul>
<p>a paragraph</p>

<script type="text/javascript">
    $("li").hasClass("yaotaiyang");
  //=> true
</script>
```

height

```
height()   => number
height(value)   => self
height(function(index, oldHeight){ ... })   => self
```

获取对象集合中第一个元素的高度；或者设置对象集合中所有元素的高度。

```
$('#foo').height()   // => 123
$(window).height()   // => 838 (viewport height)
$(document).height() // => 22302
```

hide

```
hide()   => self
```

Hide elements in this collection by setting their `display` CSS property to `none`.

通过设置css的属性 `display` 为 `none` 来将对象集合中的元素隐藏。

html

```
html()   => string
html(content)   => self
html(function(index, oldHtml){ ... })   => self
```

获取或设置对象集合中元素的HTML内容。当没有给定content参数时，返回对象集合中第一个元素的innerHtml。当给定content参数时，用其替换对象集合中每个元素的内容。content可以是append中描述的所有类型。

```
// autolink everything that looks like a Twitter username
$('.comment p').html(function(idx, oldHtml){
  return oldHtml.replace(/(^|\W)@(\w{1,15})/g,
    '$1@<a href="http://twitter.com/$2">$2</a>')
})
```

index

```
index([element])   => number
```

获取一个元素的索引值（愚人码头注：从0开始计数）。当elemen参数没有给出时，返回当前元素在兄弟节点中的位置。当element参数给出时，返回它在当前对象集合中的位置。如果没有找到该元素，则返回 `-1`。

```
$('li:nth-child(2)').index()  //=> 1
```

indexOf

```
indexOf(element, [fromIndex])   => number
```

Get the position of an element in the current collection. If fromIndex number is given, search only from that position onwards. Returns the 0-based position when found and `-1` if not found. Use of index is recommended over this method.

在当前对象集合中获取一个元素的索引值（愚人码头注：从0开始计数）。如果给定formindex参数，从该位置开始往后查找，返回基于0的索引值，如果没找到，则返回 `-1` 。index 方法是基于这个方法实现的。

这是一个Zepto的方法，不是jquer的api。

insertAfter

```
insertAfter(target)   => self
```

将集合中的元素插入到指定的目标元素后面（愚人码头注：外部插入）。这个有点像 after，但是使用方式相反。

```
$('<p>Emphasis mine.</p>').insertAfter('blockquote')
```

insertBefore

```
insertBefore(target)   => self
```

将集合中的元素插入到指定的目标元素前面（愚人码头注：外部插入）。这个有点像 before，但是使用方式相反。

```
$('<p>See the following table:</p>').insertBefore('table')
```

is

```
is(selector)   => boolean
```

判断当前元素集合中的第一个元素是否符css选择器。对于基础支持jquery的非标准选择器类似： `:visible` 包含在可选的“selector”模块中。

jQuery CSS extensions 不被支持。 选择“selector”模块仅仅能支持有限几个最常用的方式。

last

```
last()   => collection
```

获取对象集合中最后一个元素。

```
$('li').last()
```

map

```
map(function(index, item){ ... })   => collection
```

遍历对象集合中的所有元素。通过遍历函数返回值形成一个新的集合对象。在遍历函数中 `this` 关键之指向当前循环的项（遍历函数中的第二个参数）。

遍历中返回 `null` 和 `undefined`，遍历将结束。

```
// get text contents of all elements in collection
elements.map(function(){ return $(this).text() }).get().join(', ')
```

next

```
next()   => collection
next(selector)   => collection v1.0+
```

Get the next sibling–optionally filtered by selector–of each element in the collection.

获取对象集合中每一个元素的下一个兄弟节点(可以选择性的带上过滤选择器)。

```
$('dl dt').next()   //=> the DD elements
```

not

```
not(selector)   => collection
not(collection)   => collection
not(function(index){ ... })   => collection
```

过滤当前对象集合，获取一个新的对象集合，它里面的元素不能匹配css选择器。如果另一个参数为Zepto对象集合，那么返回的新Zepto对象中的元素都不包含在该参数对象中。如果参数是一个函数。仅仅包含函数执行为false值得时候的元素，函数的 `this` 关键字指向当前循环元素。

与它相反的功能，查看 filter.

offset

```
offset()   => object
offset(coordinates)   => self v1.0+
offset(function(index, oldOffset){ ... })   => self v1.0+
```

获得当前元素相对于document的位置。返回一个对象含有： `top` , `left` , `width` 和 `height`

当给定一个含有 `left` 和 `top` 属性对象时，使用这些值来对集合中每一个元素进行相对于document的定位。

offsetParent v1.0+

```
offsetParent()   => collection
```

找到第一个定位过的祖先元素，意味着它的css中的 `position` 属性值为“relative”, “absolute” or “fixed”

parent

```
parent([selector])   => collection
```

获取对象集合中每个元素的直接父元素。如果css选择器参数给出。过滤出符合条件的元素。

parents

```
parents([selector])   => collection
```

获取对象集合每个元素所有的祖先元素。如果css选择器参数给出，过滤出符合条件的元素。

如果想获取直接父级元素，使用 parent。如果只想获取到第一个符合css选择器的元素，使用closest。

```
$('h1').parents()   //=> [<div#container>, <body>, <html>]
```

pluck

```
pluck(property)   => array
```

获取对象集合中每一个元素的属性值。返回值为 `null` 或 `undefined` 值得过滤掉。

```
$('body > *').pluck('nodeName') // => ["DIV", "SCRIPT"]

// implementation of Zepto's `next` method
$.fn.next = function(){
  return $(this.pluck('nextElementSibling'))
}
```

这是一个Zepto的方法，不是jquery的api

position v1.0+

```
position()   => object
```

获取对象集合中第一个元素的位置。相对于 offsetParent。当绝对定位的一个元素靠近另一个元素的时候，这个方法是有用的。

Returns an object with properties: `top` , `left` .

```
var pos = element.position()

// position a tooltip relative to the element
$('#tooltip').css({
  position: 'absolute',
  top: pos.top - 30,
  left: pos.left
})
```

prepend

```
prepend(content)   => self
```

将参数内容插入到每个匹配元素的前面（愚人码头注：元素内部插入）。插入d的元素可以试html字符串片段，一个dom节点，或者一个节点的数组。

```
$('ul').prepend('<li>first list item</li>')
```

prependTo

```
prependTo(target)   => self
```

将所有元素插入到目标前面（愚人码头注：元素内部插入）。这有点像prepend，但是是相反的方式。

```
$('<li>first list item</li>').prependTo('ul')
```

prev

```
prev()   => collection
prev(selector)   => collection v1.0+
```

获取对象集合中每一个元素的前一个兄弟节点，通过选择器来进行过滤。

prop v1.0+

```
prop(name)   => value
prop(name, value)   => self
prop(name, function(index, oldValue){ ... })   => self
```

读取或设置dom元素的属性值。它在读取属性值的情况下优先于 attr，因为这些属性值会因为用户的交互发生改变，如 `checked` 和 `selected` 。

简写或小写名称，比如 `for` , `class` , `readonly` 及类似的属性，将被映射到实际的属性上，比如 `htmlFor` , `className` , `readOnly` , 等等。

push

```
push(element, [element2, ...])   => self
```

Add elements to the end of the current collection.

添加元素到当前对象集合的最后。

这是一个zepto的方法，不是jquery的api

ready

```
ready(function($){ ... })   => self
```

添加一个事件侦听器，当页面DOM加载完毕 “DOMContentLoaded” 事件触发时触发。建议使用 $())来代替这种用法。

reduce

```
reduce(function(memo, item, index, array){ ... }, [initial])   => value
```

与 Array.reduce有相同的用法，遍历当前对象集合。memo是函数上次的返回值。迭代进行遍历。

这是一个zepto的方法，不是jquery的api

remove

```
remove()   => self
```

从其父节点中删除当前集合中的元素，有效的从dom中移除。

removeAttr

```
removeAttr(name)   => self
```

移除当前对象集合中所有元素的指定属性。

removeClass

```
removeClass([name])   => self
removeClass(function(index, oldClassName){ ... })   => self
```

移除当前对象集合中所有元素的指定class。如果没有指定name参数，将移出所有的class。多个class参数名称可以利用空格分隔。下例移除了两个class。

```
<input class="taiyang yueliang" id="check1" type="checkbox" checked="checked">
<input class="yaotaiyang" id="check2" type="checkbox">

<script type="text/javascript">
    $("#check1").removeClass("taiyang yueliang")
    //=>[<input class id="check1" type="checkbox" checked="checked">]
</script>
```

replaceWith

```
replaceWith(content)   => self
```

用给定的内容替换所有匹配的元素。(包含元素本身)。content参数可以为 before中描述的类型。

scrollLeft v1.1+

```
scrollLeft()   => number
scrollLeft(value)   => self
```

获取或设置页面上的滚动元素或者整个窗口向右滚动的像素值。

scrollTop v1.0+

```
scrollTop()   => number
scrollTop(value)   => self v1.1+
```

获取或设置页面上的滚动元素或者整个窗口向下滚动的像素值。

show

```
show()   => self
```

恢复对象集合中每个元素默认的“display”值。如果你用 hide将元素隐藏，用该属性可以将其显示。相当于去掉了 `display：none` 。

siblings

```
siblings([selector])   => collection
```

获取对象集合中所有元素的兄弟节点。如果给定CSS选择器参数，过滤出符合选择器的元素。

size

```
size()   => number
```

获取对象集合中元素的数量。

slice

```
slice(start, [end])   => array
```

提取这个数组 `array` 的子集，从 `start` 开始，如果给定 `end`，提取从从 `start` 开始到 `end` 结束的元素，但是不包含 `end` 位置的元素。

text

```
text()   => string
text(content)   => self
text(function(index, oldText){ ... })   => self v1.1.4+
```

获取或者设置所有对象集合中元素的文本内容。当没有给定content参数时，返回当前对象集合中第一个元素的文本内容（包含子节点中的文本内容）。当给定content参数时，使用它替换对象集合中所有元素的文本内容。它有待点似 html，与它不同的是它不能用来获取或设置HTML。

toggle

```
toggle([setting])   => self
```

显示或隐藏匹配元素。如果 `setting` 为true，相当于show 法。如果 `setting` 为false。相当于hide方法。

```
var input = $('input[type=text]')
$('#too_long').toggle(input.val().length > 140)
```

toggleClass

```
toggleClass(names, [setting])   => self
toggleClass(function(index, oldClassNames){ ... }, [setting])   => self
```

在匹配的元素集合中的每个元素上添加或删除一个或多个样式类。如果class的名称存在则删除它，如果不存在，就添加它。如果 `setting` 的值为真，这个功能类似于 addClass，如果为假，这个功能类似与 removeClass。

unwrap

```
unwrap()   => self
```

移除集合中每个元素的直接父节点，并把他们的子元素保留在原来的位置。 基本上，这种方法删除上一的祖先元素，同时保持DOM中的当前元素。

```
$(document.body).append('<div id=wrapper><p>Content</p></div>')
$('#wrapper p').unwrap().parents()  //=> [<body>, <html>]
```

val

```
val()   => string
val(value)   => self
val(function(index, oldValue){ ... })   => self
```

获取或设置匹配元素的值。当没有给定value参数，返回第一个元素的值。如果是 `<select multiple>` 标签，则返回一个数组。当给定value参数，那么将设置所有元素的值。

width

```
width()   => number
width(value)   => self
width(function(index, oldWidth){ ... })   => self
```

获取对象集合中第一个元素的宽；或者设置对象集合中所有元素的宽。

```
$('#foo').width()   // => 123
$(window).width()   // => 768 (viewport width)
$(document).width() // => 768
```

wrap

```
wrap(structure)   => self
wrap(function(index){ ... })   => self v1.0+
```

在每个匹配的元素外层包上一个html元素。structure参数可以是一个单独的元素或者一些嵌套的元素。也可以是一个html字符串片段或者dom节点。还可以是一个生成用来包元素的回调函数，这个函数返回前两种类型的包裹片段。

需要提醒的是：该方法对于dom中的节点有着很好的支持。如果将 `wrap()` 用在一个新的元素上，然后再将结果插入到document中，此时该方法无效。

```
// wrap each button in a separate span:
$('.buttons a').wrap('<span>')

// wrap each code block in a div and pre:
$('code').wrap('<div class=highlight><pre /></div>')

// wrap all form inputs in a span with classname
// corresponding to input type:
$('input').wrap(function(index){
  return '<span class=' + this.type + 'field />'
})
//=> <span class=textfield><input type=text /></span>,
//   <span class=searchfield><input type=search /></span>

// WARNING: will not work as expected!
$('<em>broken</em>').wrap('<li>').appendTo(document.body)
// do this instead:
$('<em>better</em>').appendTo(document.body).wrap('<li>')
```

wrapAll

```
wrapAll(structure)   => self
```

在所有匹配元素外面包一个单独的结构。结构可以是单个元素或 几个嵌套的元素，并且可以通过在作为HTML字符串或DOM节点。

```
// wrap all buttons in a single div:
$('a.button').wrapAll('<div id=buttons />')
```

wrapInner

```
wrapInner(structure)   => self
wrapInner(function(index){ ... })   => self v1.0+
```

将每个元素中的内容包裹在一个单独的结构中。结构可以是单个元件或多个嵌套元件，并且可以通过在作为HTML字符串或DOM节点，或者是一个生成用来包元素的回调函数，这个函数返回前两种类型的包裹片段。

```
// wrap the contents of each navigation link in a span:
$('nav a').wrapInner('<span>')

// wrap the contents of each list item in a paragraph and emphasis:
$('ol li').wrapInner('<p><em /></p>')
```

Detect methods

Detect module

该检测方法可以在不同的环境中微调你的站点或者应用程序，并帮助你识别手机和平板；以及不同的浏览器和操作系统。

```
// The following boolean flags are set to true if they apply,
// if not they're either set to `false` or `undefined`.
// We recommend accessing them with `!!` prefixed to coerce to a boolean.

// general device type
$.os.phone
$.os.tablet

// specific OS
$.os.ios
$.os.android
$.os.webos
$.os.blackberry
$.os.bb10
$.os.rimtabletos

// specific device type
$.os.iphone
$.os.ipad
$.os.ipod // [v1.1]
$.os.touchpad
$.os.kindle

// specific browser
$.browser.chrome
$.browser.firefox
$.browser.safari // [v1.1]
$.browser.webview // (iOS) [v1.1]
$.browser.silk
$.browser.playbook
$.browser.ie // [v1.1]

// 此外，版本信息是可用的。
// 下面是运行??iOS 6.1的iPhone所返回的。
!!$.os.phone         // => true
!!$.os.iphone        // => true
!!$.os.ios           // => true
$.os.version       // => "6.1"
$.browser.version  // => "536.26"
```

事件处理

$.Event

```
$.Event(type, [properties])   => event
```

创建并初始化一个指定的DOM事件。如果给定properties对象，使用它来扩展出新的事件对象。默认情况下，事件被设置为冒泡方式；这个可以通过设置 `bubbles` 为 `false` 来关闭。

一个事件初始化的函数可以使用 trigger来触发。

```
$.Event('mylib:change', { bubbles: false })
```

$.proxy v1.0+

```
$.proxy(fn, context)   => function
$.proxy(fn, context, [additionalArguments...])   => function v1.1.4+
$.proxy(context, property)   => function
$.proxy(context, property, [additionalArguments...])   => function v1.1.4+
```

接受一个函数，然后返回一个新函数，并且这个新函数始终保持了特定的上下文(context)语境，新函数中 `this` 指向context参数。另外一种形式，原始的function是从上下文(context)对象的特定属性读取。

如果传递超过2个的额外参数，它们被用于 传递给fn参数的函数 引用。

```
var obj = {name: 'Zepto'},
    handler = function(){ console.log("hello from + ", this.name) }

// ensures that the handler will be executed in the context of `obj`:
$(document).on('click', $.proxy(handler, obj))
```

bind

不推荐, 使用 on 代替。

```
bind(type, function(e){ ... })   => self
bind(type, [data], function(e){ ... })   => self v1.1+
bind({ type: handler, type2: handler2, ... })   => self
bind({ type: handler, type2: handler2, ... }, [data])   => self v1.1+
```

为一个元素绑定一个处理事件。

delegate

不推荐, 使用 on 代替。

```
delegate(selector, type, function(e){ ... })   => self
delegate(selector, { type: handler, type2: handler2, ... })   => self
```

基于一组特定的根元素为所有选择器匹配的元素附加一个处理事件，匹配的元素可能现在或将来才创建。

die

不推荐, 使用 on 代替。

```
die(type, function(e){ ... })   => self
die({ type: handler, type2: handler2, ... })   => self
```

删除通过 live 添加的事件。

event.isDefaultPrevented v1.1+

```
event.isDefaultPrevented()   => boolean
```

如果 `preventDefault()` 被该事件的实例调用，那么返回true。 这可作为跨平台的替代原生的 `defaultPrevented` 属性，如果 `defaultPrevented` 缺失或在某些浏览器下不可靠的时候。

```
// trigger a custom event and check whether it was cancelled
var event = $.Event('custom')
element.trigger(event)
event.isDefaultPrevented()
```

event.isImmediatePropagationStopped v1.1+

```
event.isImmediatePropagationStopped()   => boolean
```

如果 `stopImmediatePropagation()` 被该事件的实例调用，那么返回true。Zepto在不支持该原生方法的浏览器中实现它， （例如老版本的Android）。

event.isPropagationStopped v1.1+

```
event.isPropagationStopped()   => boolean
```

如果 `stopPropagation()` 被该事件的实例调用，那么返回true。

live

不推荐, 使用 on 代替。

```
live(type, function(e){ ... })   => self
live({ type: handler, type2: handler2, ... })   => self
```

类似delegate，添加一个个事件处理器到符合目前选择器的所有元素匹配，匹配的元素可能现在或将来才创建。

off

```
off(type, [selector], function(e){ ... })   => self
off({ type: handler, type2: handler2, ... }, [selector])   => self
off(type, [selector])   => self
off()   => self
```

移除通过 on 添加的事件.移除一个特定的事件处理程序， 必须通过用 `on()` 添加的那个相同的函数。否则，只通过事件类型调用此方法将移除该类型的所有处理程序。如果没有参数，将移出当前元素上全部的注册事件。

on

```
on(type, [selector], function(e){ ... })   => self
on(type, [selector], [data], function(e){ ... })   => self v1.1+
on({ type: handler, type2: handler2, ... }, [selector])   => self
on({ type: handler, type2: handler2, ... }, [selector], [data])   => self v1.1+
```

添加事件处理程序到对象集合中得元素上。多个事件可以通过空格的字符串方式添加，或者以事件类型为键、以函数为值的对象 方式。如果给定css选择器，当事件在匹配该选择器的元素上发起时，事件才会被触发（愚人码头注：即事件委派，或者说事件代理）。

如果给定 `data` 参数，这个值将在事件处理程序执行期间被作为有用的 `event.data` 属性

事件处理程序在添加该处理程序的元素、或在给定选择器情况下匹配该选择器的元素的上下文中执行(愚人码头注：this指向触发事件的元素)。 当一个事件处理程序返回 `false`， `preventDefault()` 和 `stopPropagation()` 被当前事件调用的情况下， 将防止默认浏览器操作，如链接。

如果 `false` 在回调函数的位置上作为参数传递给这个方法， 它相当于传递一个函数，这个函数直接返回 `false`。（愚人码头注：即将 `false` 当作 `function(e){ ... }` 的参数，作为 `function(){ return false; }` 的简写形式，例如： `$("a.disabled").on("click", false);` 这相当于 `$("a.disabled").on("click", function(){ return false; } );` ）

```
var elem = $('#content')
// observe all clicks inside #content:
elem.on('click', function(e){ ... })
// observe clicks inside navigation links in #content
elem.on('click', 'nav a', function(e){ ... })
// all clicks inside links in the document
$(document).on('click', 'a', function(e){ ... })
// disable following any navigation link on the page
$(document).on('click', 'nav a', false)
```

one

```
one(type, [selector], function(e){ ... })   => self
one(type, [selector], [data], function(e){ ... })   => self v1.1+
one({ type: handler, type2: handler2, ... }, [selector])   => self
one({ type: handler, type2: handler2, ... }, [selector], [data])   => self v1.1+
```

添加一个处理事件到元素，当第一次执行事件以后，该事件将自动解除绑定，保证处理函数在每个元素上最多执行一次。 `selector` 和 `data` 等参数说明请查看 `.on()` 。

trigger

```
trigger(event, [args])   => self
```

在对象集合的元素上触发指定的事件。事件可以是一个字符串类型，也可以是一个 通过 $.Event 定义的事件对象。如果给定args参数，它会作为参数传递给事件函数。

```
// add a handler for a custom event
$(document).on('mylib:change', function(e, from, to){
  console.log('change on %o with data %s, %s', e.target, from, to)
})
// trigger the custom event
$(document.body).trigger('mylib:change', ['one', 'two'])
```

Zepto仅仅支持在dom元素上触发事件。

triggerHandler

```
triggerHandler(event, [args])   => self
```

像 trigger，它只在当前元素上触发事件，但不冒泡。

```
 $("input").triggerHandler('focus');
        // 此时input上的focus事件触发，但是input不会获取焦点
        $("input").trigger('focus');
        // 此时input上的focus事件触发，input获取焦点
```

unbind

Deprecated, use off instead.

```
unbind(type, function(e){ ... })   => self
unbind({ type: handler, type2: handler2, ... })   => self
```

移除通过 bind 注册的事件。

undelegate

Deprecated, use off instead.

```
undelegate(selector, type, function(e){ ... })   => self
undelegate(selector, { type: handler, type2: handler2, ... })   => self
```

移除通过delegate 注册的事件。

Ajax 请求

$.ajax

```
$.ajax(options)   => XMLHttpRequest
```

执行Ajax请求。它可以是本地资源，或者通过支持HTTP access control的浏览器 或者通过JSONP来实现跨域。

选项:

- `type` (默认： “GET”)：请求方法 (“GET”, “POST”, or other)
- `url` (默认： 当前地址)：发送请求的地址
- `data` (默认：none)：发送到服务器的数据；如果是GET请求，它会自动被作为参数拼接到url上。非String对象将通过 $.param 得到序列化字符串。
- `processData` (默认： true)： 对于非Get请求。是否自动将 `data` 转换为字符串。
- `contentType` (默认： “application/x-www-form-urlencoded”)： 发送信息至服务器时内容编码类型。 (这也可以通过设置 `headers`)。通过设置 `false` 跳过设置默认值。
- `mimeType` (默认： none): 覆盖响应的MIME类型。 v1.1+
- `dataType` (默认： none)：预期服务器返回的数据类型(“json”, “jsonp”, “xml”, “html”, or “text”)
- `jsonp` (默认：“callback”): JSONP回调查询参数的名称
- `jsonpCallback` (默认： “jsonp{N}”): 全局JSONP回调函数的 字符串（或返回的一个函数）名。设置该项能启用浏览器的缓存。 v1.1+
- `timeout` (默认： `0`): 以毫秒为单位的请求超时时间, `0` 表示不超时。
- `headers` : Ajax请求中额外的HTTP信息头对象
- `async` (默认：true): 默认设置下，所有请求均为异步。如果需发送同步请求，请将此设置为 `false` 。
- `global` (默认：true): 请求将触发全局Ajax事件处理程序，设置为 false 将不会触发全局 Ajax 事件。
- `context` (默认：window): 这个对象用于设置Ajax相关回调函数的上下文(this指向)。
- `traditional` (默认： false): 激活传统的方式通过$.param来得到序列化的 `data` 。
- `cache` (默认： true): 浏览器是否应该被允许缓存GET响应。从v1.1.4开始，当dataType选项为 `"script"` 或 `jsonp` 时，默认为 `false` 。
- `xhrFields` (默认： none): 一个对象包含的属性被逐字复制到XMLHttpRequest的实例。 v1.1+
- `username` & `password` (默认： none): HTTP基本身份验证凭据。 v1.1+

如果URL中含有 `=?` 或者 `dataType` 是“jsonp”，这讲求将会通过注入一个 `<script>` 标签来代替使用 XMLHttpRequest (查看 JSONP)。此时 `contentType` , `dataType` , `headers` 有限制， `async` 不被支持。

Ajax 回调函数

你可以指定以下的回调函数，他们将按给定的顺序执行：

1. `beforeSend(xhr, settings)` ：请求发出前调用，它接收xhr对象和settings作为参数对象。如果它返回 `false` ，请求将被取消。
2. `success(data, status, xhr)` ：请求成功之后调用。传入返回后的数据，以及包含成功代码的字符串。
3. `error(xhr, errorType, error)` ：请求出错时调用。 (超时，解析错误，或者状态码不在HTTP 2xx)。
4. `complete(xhr, status)` ：请求完成时调用，无论请求失败或成功。

Promise 回调接口 v1.1+

如果可选的“callbacks” 和 “deferred” 模块被加载，从 `$.ajax()` 返回的XHR对象实现了promise接口链式的回调：

```
xhr.done(function(data, status, xhr){ ... })
xhr.fail(function(xhr, errorType, error){ ... })
xhr.always(function(){ ... })
xhr.then(function(){ ... })
```

这些方法取代了 `success` , `error` , 和 `complete` 回调选项.

Ajax 事件

当 `global: true` 时。在Ajax请求生命周期内，以下这些事件将被触发。

1. `ajaxStart` *(global)*：如果没有其他Ajax请求当前活跃将会被触发。
2. `ajaxBeforeSend` (data: xhr, options)：再发送请求前，可以被取消。
3. `ajaxSend` (data: xhr, options)：像 `ajaxBeforeSend` ，但不能取消。
4. `ajaxSuccess` (data: xhr, options, data)：当返回成功时。
5. `ajaxError` (data: xhr, options, error)：当有错误时。
6. `ajaxComplete` (data: xhr, options)：请求已经完成后，无论请求是成功或者失败。

7. `ajaxStop` *(global)*：如果这是最后一个活跃着的Ajax请求，将会被触发。

默认情况下，Ajax事件在document对象上触发。然而，如果请求的 `context` 是一个DOM节点，该事件会在此节点上触发然后再DOM中冒泡。唯一的例外是 `ajaxStart` & `ajaxStop` 这两个全局事件。

```
$(document).on('ajaxBeforeSend', function(e, xhr, options){
  // This gets fired for every Ajax request performed on the page.
  // The xhr object and $.ajax() options are available for editing.
  // Return false to cancel this request.
})

$.ajax({
  type: 'GET',
  url: '/projects',
  // data to be added to query string:
  data: { name: 'Zepto.js' },
  // type of data we are expecting in return:
  dataType: 'json',
  timeout: 300,
  context: $('body'),
  success: function(data){
    // Supposing this JSON payload was received:
    //   {"project": {"id": 42, "html": "<div>..." }}
    // append the HTML to context object.
    this.append(data.project.html)
  },
  error: function(xhr, type){
    alert('Ajax error!')
  }
})

// post a JSON payload:
$.ajax({
  type: 'POST',
  url: '/projects',
  // post payload:
  data: JSON.stringify({ name: 'Zepto.js' }),
  contentType: 'application/json'
})
```

$.ajaxJSONP

不推荐, 使用 $.ajax 代替。

```
$.ajaxJSONP(options)   => mock XMLHttpRequest
```

执行JSONP跨域获取数据。

此方法相对 $.ajax 没有优势，建议不要使用。

$.ajaxSettings

一个包含Ajax请求的默认设置的对象。大部分的设置在 $.ajax中已经描述。以下设置为全局非常有用：

- `timeout` (默认： `0`)：对Ajax请求设置一个非零的值指定一个默认的超时时间，以毫秒为单位。
- `global` (默认： true)：设置为false。以防止触发Ajax事件。
- `xhr` (默认：XMLHttpRequest factory)：设置为一个函数，它返回XMLHttpRequest实例(或一个兼容的对象)
- `accepts`：从服务器请求的MIME类型，指定`dataType`值：
    - script: “text/javascript, application/javascript”
    - json: “application/json”
    - xml: “application/xml, text/xml”
    - html: “text/html”
    - text: “text/plain”

$.get

```
$.get(url, function(data, status, xhr){ ... })   => XMLHttpRequest
$.get(url, [data], [function(data, status, xhr){ ... }], [dataType])   => XMLHttpRequest
```

执行一个Ajax GET请求。这是一个 $.ajax的简写方式。

```
$.get('/whatevs.html', function(response){
  $(document.body).append(response)
})
```

$.getJSON

```
$.getJSON(url, function(data, status, xhr){ ... })   => XMLHttpRequest
$.getJSON(url, [data], function(data, status, xhr){ ... })   => XMLHttpRequest v1.0+
```

通过 Ajax GET请求获取JSON数据。这是一个 $.ajax 的简写方式。

```
$.getJSON('/awesome.json', function(data){
  console.log(data)
})

// fetch data from another domain with JSONP
$.getJSON('//example.com/awesome.json?callback=?', function(remoteData){
  console.log(remoteData)
})
```

$.param

```
$.param(object, [shallow])   => string
$.param(array)   => string
```

序列化一个对象，在Ajax请求中提交的数据使用URL编码的查询字符串表示形式。如果shallow设置为true。嵌套对象不会被序列化，嵌套数组的值不会使用放括号在他们的key上。

如果任何对象的某个属性值是一个函数，而不是一个字符串，该函数将被调用并且返回值后才会被序列化。

此外，还接受 serializeArray格式的数组，其中每个项都有 “name” 和 “value”属性。

```
$.param({ foo: { one: 1, two: 2 }})
//=> "foo[one]=1&foo[two]=2)"

$.param({ ids: [1,2,3] })
//=> "ids[]=1&ids[]=2&ids[]=3"

$.param({ ids: [1,2,3] }, true)
//=> "ids=1&ids=2&ids=3"

$.param({ foo: 'bar', nested: { will: 'not be ignored' }})
//=> "foo=bar&nested[will]=not+be+ignored"

$.param({ foo: 'bar', nested: { will: 'be ignored' }}, true)
//=> "foo=bar&nested=[object+Object]"

$.param({ id: function(){ return 1 + 2 } })
//=> "id=3"
```

$.post

```
$.post(url, [data], function(data, status, xhr){ ... }, [dataType])   => XMLHttpRequest
```

执行Ajax POST请求。这是一个 $.ajax 的简写方式。

```
$.post('/create', { sample: 'payload' }, function(response){
  // process response
})
```

`data` 参数可以是一个字符串：

```
$.post('/create', $('#some_form').serialize(), function(response){
  // ...
})
```

load

```
load(url, function(data, status, xhr){ ... })   => self
```

通过GET Ajax载入远程 HTML 内容代码并插入至 当前的集合 中。另外，一个css选择器可以在url中指定，像这样，可以使用匹配selector选择器的HTML内容来更新集合。

```
$('#some_element').load('/foo.html #bar')
```

如果没有给定CSS选择器，将使用完整的返回文本。

请注意，在没有选择器的情况下，任何javascript块都会执行。如果带上选择器，匹配选择器内的script将会被删除。

表单方法

serialize

```
serialize()   => string
```

在Ajax post请求中将用作提交的表单元素的值编译成 URL编码的 字符串。

serializeArray

```
serializeArray()   => array
```

将用作提交的表单元素的值编译成拥有 `name` 和 `value` 对象组成的数组。不能使用的表单元素，buttons，未选中的radio buttons/checkboxs 将会被跳过。结果不包含file inputs的数据。

```
$('form').serializeArray()
//=> [{ name: 'size', value: 'micro' },
//    { name: 'name', value: 'Zepto' }]
```

submit

```
submit()   => self
submit(function(e){ ... })   => self
```

为 "submit" 事件绑定一个处理函数，或者触发元素上的 "submit" 事件。当没有给定function参数时，触发当前表单“submit”事件，并且执行默认的提交表单行为，除非调用了 `preventDefault()` 。

当给定function参数时，在当前元素上它简单得为其在“submit”事件绑定一个处理函数。

Effects

$.fx

全局地动画设置：

- `$.fx.off` (在支持css transition 的浏览器中默认为false)：设置true来禁止所有 `animate()` transitions。
- `$.fx.speeds` ：用来设置动画时间的对象。
  - `_default` (400 ms)
  - `fast` (200 ms)
  - `slow` (600 ms)

改变现有值或者添加一个新属性去影响使用一个字符串来设置时间的动画。

animate

```
animate(properties, [duration, [easing, [function(){ ... }]]])   => self
      animate(properties, { duration: msec, easing: type, complete: fn })   => self
      animate(animationName, { ... })   => self
```

对当前对象集合中元素进行css transition属性平滑过渡。

- `properties`：一个对象，该对象包含了css动画的值，或者css帧动画的名称。
- `duration` (默认 400)：以毫秒为单位的时间，或者一个字符串。
  - `fast` (200 ms)
  - `slow` (600 ms)
  - 任何 `$.fx.speeds` 自定义属性
- `easing` (默认 `linear`)：指定动画的缓动类型，使用以下一个：
  - `ease`
  - `linear`
  - `ease-in` / `ease-out`
  - `ease-in-out`
  - `cubic-bezier(...)`
- `complete` ：动画完成时的回调函数

`delay`

Zepto 还支持以下 CSS transform 属性：

- `translate(X|Y|Z|3d)`
- `rotate(X|Y|Z|3d)`
- `scale(X|Y|Z)`
- `matrix(3d)`
- `perspective`
- `skew(X|Y)`

如果duration参数为 `0` 或 `$.fx.off` 为 true(在不支持css transitions的浏览器中默认为true)，动画将不被执行；替代动画效果的目标位置会即刻生效。类似的，如果指定的动画不是通过动画完成，而且动画的目标位置即可生效。这种情况下没有动画， `complete` 方法也不会被调用。

如果第一个参数是字符串而不是一个对象，它将被当作一个css关键帧动画 CSS keyframe animation的名称。

```
$("#some_element").animate({
        opacity: 0.25,
        left:
        '50px',
        color:
        '#abcdef',
        rotateZ:
        '45deg',
        translate3d: '0,10px,0'
        }, 500,
        'ease-out')
```

Zepto只使用css过渡效果的动画。jquery的easings不会支持。jquery的相对变化("=+10px") syntax 也不支持。请查看 list of animatable properties。浏览器的支持可能不同，所以一定要测试你所想要支持的浏览器。

Touch

Touch events

“touch”模块添加以下事件，可以使用 on 和 off。

- `tap` —元素tap的时候触发。
- `singleTap` and `doubleTap` — 这一对事件可以用来检测元素上的单击和双击。(如果你不需要检测单击、双击，使用 `tap` 代替)。
- `longTap` — 当一个元素被按住超过750ms触发。
- `swipe`, `swipeLeft`, `swipeRight`, `swipeUp`, `swipeDown` — 当元素被划过时触发。(可选择给定的方向)

这些事件也是所有Zepto对象集合上的快捷方法。

```
<style>.delete { display: none; }</style>

<ul id=items>
  <li>List item 1 <span class=delete>DELETE</span></li>
  <li>List item 2 <span class=delete>DELETE</span></li>
</ul>

<script>
// show delete buttons on swipe
$('#items li').swipe(function(){
  $('.delete').hide()
  $('.delete', this).show()
})

// delete row on tapping delete button
$('.delete').tap(function(){
  $(this).parent('li').remove()
})
</script>
```

Change Log

v1.1.0 — *05 Dec 2013* — diff

Notable changes

- IE10+ support
- Huge speed optimizations for simple CSS selectors (classname, ID) and DOM element creation
- Provide `$.Callbacks` and `$.Deferred` in optional modules
- Removed `fx` and `detect` modules from default build

Ajax

- New supported `$.ajax()` options:
  - `xhrFields`
  - `mimeType`
  - `jsonpCallback`
  - `username` & `password`
- Promise interface supported when loading the optional “callbacks” and “deferred” modules:
  - `xhr.done(function(data, status, xhr){ ... })`
  - `xhr.fail(function(xhr, errorType, error){ ... })`
  - `xhr.always(function(){ ... })`
- Enable mutating Ajax settings in the `beforeSend` callback
- Fix JSONP callbacks for errored responses on Android
- Ensure consistent `Accept` request HTTP header across browsers
- Fix `$.param()` for jQuery compatibility when handling complex nested objects
- Support IIS JavaScript MIME type
- Pass “abort” and “timeout” status to global `ajaxError` event handlers

Event

- Provide `isDefaultPrevented()`, `stopImmediatePropagation()`, and related methods for all events
- Support the `data` argument in `.bind()`, `.on()`, and `.one()`
- Support CSS selector argument in `.one()` for event delegation
- Support `.on('ready')` as an alias for `.ready()`

- Enable event handlers on plain old JS objects
- Many fixes related to event delegation

Data

- Cleanup `.data()` values on DOM element removal with `.remove/empty()`
- `.data()` now assumes that numbers that begin with zeroes are strings
- `.removeData()` (no argument) now removes all data on the element
- Enable reading `data-*` attributes that have underscores in the name

Misc.

- Support simple DOM property names in `.prop(name)` such as `for`, `class`, `readonly`…
- Implement the `.scrollLeft([value])` method
- Support setting `.scrollTop(value)`
- Fix `$(document).width/height()`
- Support fetching multiple CSS values via array in `.css(['prop1', 'prop2', ...])`
- Support setting CSS transition delay via `delay` option for `.animate()`
- Ensure that `.animate()` callback always firesParty like it's one-oh!_

v1.0 — *02 Mar 2013* — diff

Party like it's one-oh!

Notable changes

- Zepto is now compatible with Twitter Bootstrap
- Portable, completely new node.js-based build system
- Fully automated tests with PhantomJS and Travis CI
- Removed `touch` module from default distribution

New features

- `$.fn.filter(function(index){ ... })`
- `$.fn.contents()`
- `$.fn.wrapInner()`
- `$.fn.scrollTop()`
- `$.contains()`
- `$.fn.has()`

- `$.fn.position()`
- `$.fn.offsetParent()`
- `$.parseJSON()`
- `$.camelCase()`
- `$.isWindow()`
- `$.grep()` (interface to `Array.filter`)
- Support `$(html, attributes)` syntax for element creation
- Emulate `mouseenter` and `mouseleave` events
- Bootstrap compat: support `$.fn.offset(coordinates)`
- Bootstrap compat: implement `$.fn.detach()`
- Add support for Ajax `cache: false` option
- Prevent scrolling when horizontal swipe events are detected
- `cancelTouch` for tap events
- `prev` and `next` now support an optional selector argument
- `$.fn.find` and `$.fn.closest` now support Zepto objects as arguments
- Enable deep copy via `$.extend(true, target, source)`
- Enable nested structures for `$.fn.wrap()` and `$.fn.wrapAll()`
- Enable function arguments for `$.fn.wrap()` and `$.fn.wrapInner()`
- Support number, boolean, JSON types in data attributes
- Support manipulating classnames on SVG elements
- Enable named durations for `animate` , e.g. `slow` .
- Support `timing-function` for `animate`
- Support event properties passed to `$.fn.trigger()` or `$.Event()`
- Selector module: support child `> *` queries
- Add detect support for mobile Chrome browser
- Add `$.os.phone` and `$.os.tablet` (booleans)
- Detect Firefox mobile, Playbooks and BB10

Fixes

- Fix passing null selector to `on` or `off`
- Fixed bug where self-closing html tags would act as open tags
- Fix `val` for multiple select
- Fix various touch and gesture bugs.
- Corrected parameters of `load` success callback to match jQuery.
- Fix `css` with 0 values and falsy values
- Fix a `css` performance issues with string values
- Fix `$.ajaxJSONP` when invoked directly
- Fix `animate` with 0 durations.
- Fix `toggle` and `fadeToggle` for multiple elements.

- Fix ajax `$.fn.load` behavior with selector
- Make `attr(name, null)` unset attribute
- Fix `animate` in Firefox
- Fix `animate` for elements just added to DOM
- Fix an escaping issue with `$.param`
- Respect `traditional: true` option in `$.ajax`
- Fix `focus` & `blur` event delegation and enable unbind
- Simple wrapping for any object passed to `$()`
- Enable `children` method for XML documents
- Don't eval `<script>` content when `src` is present
- Support `processData` option for `$.ajax()`
- Enable passing `contentType: false` to `$.ajax()`
- Apply `focus()` and `blur()` to all elements in collection
- Change `$.fn.map()` to return a Zepto collection
- Selector argument for `on(evt, selector, fn)` can be false
- Don't raise error on `$('#')`
- Provide empty object in `$.support`
- `return false` in event handler calls stopPropagation()
- Fix `$.isPlainObject()` for `window` in Opera
- `$.ajax` error callback correctly reports `abort` status
- Fix `hasClass` in collections of multiple elements
- Stop iteration in `each()` when the callback returns false
- Add ability to set `xhr` factory per-request
- Have `get()` method accept negative index
- Support for multiple class names in `toggleClass()`
- Fix error callbacks for `ajaxJSONP`
- Support optional `data` argument for various Ajax methods
- Fix DOM insertion operators for null values
- Fix dataType being set for `$.getJSON`

v1.0rc1 — *09 Apr 2012* — diff

The *semicolon-free* edition! That's right, we removed all trailing semicolons from the source and tests. They were never needed anyway.

New methods:

- clone
- prop
- $.isPlainObject
- $.inArray

- $.trim
- $.proxy

New module:

- “selector.js” with experimental support for jQuery CSS pseudo-selectors such as `:visible` and `:first`

Improvements in core:

- added missing methods for Ember.js compatibility
- improved creating DOM fragments from HTML with $())
- enable append & family to accept multiple arguments
- fix $.each context
- fix calling get without index
- fix calling val on empty collection
- using `css(property, '')` removes the property
- fix filter, is, and closest when operating on nodes that are detached from the document
- remove `end` & `andSelf` from core to the new “stack.js” plugin
- exposed important internal Zepto functions through the `$.zepto` object for extending or overriding Zepto functionality.
- data method returns undefined when there is no data
- support camelized names in data method

Apart from improving the basic `data` method in core, the “data.js” module got improvements as well:

- better jQuery compatibility
- ability to store functions
- new `removeData` method

Ajax:

- have correct ajaxComplete argument order for JSONP abort and timeout
- JSONP requests that hit a 404 will now correctly invoke the error callback
- add support for `dataType: 'jsonp'` in $.ajax
- add support for `data` in $.ajaxJSONP
- HTTP 304 status is treated as success instead of an error
- made load more compatible with jQuery
- allow Content-Type to be set via request headers
- respect Content-Type of the response if `dataType` isn’t set
- work around Chrome CORS bug when data is empty

Changes in other modules:

- fix animate for edge cases such as when there is an animation within an animated element, and improve handling of transition CSS properties
- new “singleTap” event
- improved “longTap” detection

0.8 — *03 Nov 2011* — diff

- CSS transitions for every browser with `animate()` method;
- unified event handling with `fn.on()` & `off()` ;
- Ajax global events & timeout support;
- performance boost for selectors.

See full release notes.

0.7 — *01 Aug 2011* — diff

- add `$.each` , `$.map` , `$.slice` ;
- add `.serializeArray()` , `.serialize()` ;
- add `.triggerHandler()` ;
- add `.wrap` , `.wrapAll` , `.unwrap` , `.width/height` setters, `.append` (and friends) improvements;
- add “longTap” event;
- `.anim()` accepts CSS transform properties;
- `return false` in event handlers cancels browser event behavior.

0.6 — *14 May 2011* — diff

- add `.add` , `.appendTo` , `.prependTo` , `.replaceWith` , `.empty` , `.submit` ;
- allow function args for `.add/.remove/.toggleClass` ;
- improvements to events and xhr.

0.5 — *01 Mar 2011* — diff

- add `.not` , `.children` , `.siblings` , `$.param` ;
- improve `.attr` & `.html` ;
- support callback for `.anim` .

0.4 — *21 Jan 2011* — diff

- JSONP;
- better `.find` , `.each` , `.closest` ;
- add `.eq` , `.size` , `.parent` , `.parents` , `.removeAttr` , `.val` ;
- support function args in `.html` , `.attr` ;
- adjacency methods now take Zepto objects.

0.3 — *17 Dec 2010* — diff

- add `.toggleClass` , `.attr` setter, `.last` , `.undelegate` , `.die` ;
- proxied event objects for event delegation;
- support `$` fragments.

0.2 — *01 Dec 2010* — diff

- now compatible with backbone.js;
- support event unbind;
- ajax posts with data.

0.1 — *26 Oct 2010*

First. Release. Ever.

Acknowledgements & Thanks

A big **Thank-You** goes out to all of our awesome Zepto.js contributors. May you all forever bask in glory.

The Zepto API is based on jQuery's Core API, which is released under the MIT license.

This documentation is based on the layout of the Backbone.js documentation, which is released under the MIT license.